FAST METHOD FOR THE FORWARD AND INVERSE MDCT IN AUDIO CODING

CROSS-REFERENCE TO RELATED APPLICATIONS

This patent application claims the benefit of the filing date of United States Provisional Patent Application Serial Nos. 60/181,271, filed February 9, 2000 and entitled "Fast Method for the Forward and Inverse MDCT in Audio Coding"; and 60/184,685, filed February 24, 2000 and entitled "Fast Method for the Forward and Inverse MDCT in Audio Coding", the entire contents of which are hereby expressly incorporated by reference.

FIELD OF THE INVENTION

The present invention relates to audio and image data compression and decompression applications. More specifically, the invention relates to a system and method for fast computation of modified discrete cosine transform (MDCT) and its inverse modified discrete cosine transform (IMDCT).

BACKGROUND OF THE INVENTION

Discrete cosine transform (DCT) is used extensively in data compression for image and speech signals. For example, the authors in N. Amed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Trans. Commun.*, vol. COM-23, pp. 90-93, Jan. 1974 [1]; and Man IK Chao and Sang UK Lee, "DCT methods for VLSI parallel Implementations," *IEEE Trans. On Acoustics, Speech and Signal Processing*, vol. 38, no. 1, pp. 121-127, January 1990 [2], the contents of which are hereby incorporated by reference, describe DCT methods for data compression. Many methods have been proposed to compute the DCT. For example, see Nam IR Cho and Sang UK Lee, "DCT Methods for VLSI Parallel Implementations, " *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol.38, no.1, pp.121-127, January 1990 [11]; and T. K. Truong, I. S. Reed, I. S. Hsu, H. C. Shyu, and H. M. Sho, "A Pipelined Design of a Fast Prime Factor DFT on a Finite Field," *IEEE Trans. Computers*, vol. 37, pp. 266-273, Mar. 1988 [12], the contents of which are hereby incorporated by reference.

Recently, Kek in C. W. Kok, "Fast Method for Computing Discrete Cosine Transform," *IEEE Trans. on Signal Processing*, vol. 45, no. 3, pp. 757-760, March 1997 [3], the contents of which are hereby incorporated by reference, suggested a new DCT method for computing the DCT of length $N = N_1 \times N_2$, where N_1 is an even number and N_2 is an odd number. In this method, the DCT with even length N is decomposed into two DCT's with length $N/2$. If $N/2$ is an odd number.

Further, it follows from Michael T. Heideman, "Computation of an Odd Length DCT from a Real-Value DFT of the Same Length," *IEEE trans. on Signal Processing*, vol. 40, no. 1, pp. 54-61, January 1992 [4], the contents of which are hereby incorporated by reference, that such a DCT of length $N/2$ can be computed by the use of the identical length discrete Fourier transform (DFT) method developed by Winograd. This method is described in Dean P. Kolba and Thomas W. Parks, "A Prime Factor FFT Method Using High-Speed Convolution," *IEEE Trans on Acoustics, Speech and Signal processing*, vol. ASSP-25, no. 4, pp. 281-294, August 1977 [5]; and S. Winogard, "On Computing the Discrete Fourier Transform," *Mathematics of Computation*, vol. 32, no. 141, pp. 175-199, January 1978. [6], the contents of which are hereby incorporated by reference.

By using the ideas in [3, and 4], a fast method can be developed to compute an 18-point DCT. In other words, the standard 18-point DCT is decomposed first into two 9-point DCT. Then, such a 9-point DCT can be implemented by Winograd's DFT method. However, computing the MDCT and its IMDCT of an even length sequence can involve an extensive computation.

Therefore, there is a need for a fast method to compute either the MDCT or IMDCT so that it can be implemented easily on either a processor, such as a digital signal processor (DSP) or computer. Most recently, the author in K. Konstantinides, "Fast Subband Filtering in MPEG Audio Coding," *IEEE Signal Processing Letters*, vol. 1, no. 2, pp. 26-28, Feb. 1994 [7], the contents of which are hereby incorporated by reference, suggested that the MDCT and IMDCT in MPEG audio coding can be converted first into the standard DCT and IDCT, respectively. Such a DCT or IDCT can be computed rapidly by the use of Lee's fast method in Byeong Gi Lee, "A New Method to Compute the Discrete Cosine Transform,"

IEEE Trans on Acoustics, Speech and Signal processing, vol. ASSP-32, no. 6, pp. 1243-1245, Dec. 1984 [8], the contents of which are hereby incorporated by reference.

SUMMARY OF THE INVENTION

The modified discrete cosine transform (MDCT) and its inverse modified discrete cosine transform (IMDCT) can involve an extensive computation, for example, in layer III of the MPEG audio coding standard. The present invention computes an 18-point DCT in a fast and efficient manner. Furthermore, using this 18-point DCT method, two new methods are developed to compute the MDCT and its IMDCT, respectively. The number of multiplications and additions needed to implement both of these two new methods are reduced substantially.

In one aspect, the invention describes a method performed by a computer for computing modified discrete cosine transfer comprising the steps of:

computing $x(k) = \begin{cases} [-y(26-k) - y(27+k)] \cdot b_k & \text{for} \quad 0 \le k \le 8 \\ [y(k-9) - y(26-k)] \cdot b_k & \text{for} \quad 9 \le k \le 17 \end{cases}$;

computing $Y'(n) = \sum_{k=0}^{17} x(k) \cos[\frac{\pi}{36}(2k+1)n]$ for $0 \le n \le 17$;

defining $Y(0) = Y'(0)/2$; and computing $Y(n) = Y'(n) - Y(n-1)$ for $1 \le n \le 17$.

In another aspect, the invention describes an MPEG encoder/decoder comprising:

means for computing $x(k) = \begin{cases} [-y(26-k) - y(27+k)] \cdot b_k & \text{for} \quad 0 \le k \le 8 \\ [y(k-9) - y(26-k)] \cdot b_k & \text{for} \quad 9 \le k \le 17 \end{cases}$;

means for computing $Y'(n) = \sum_{k=0}^{17} x(k) \cos[\frac{\pi}{36}(2k+1)n]$ for $0 \le n \le 17$;

means for defining $Y(0) = Y'(0)/2$; and means for computing $Y(n) = Y'(n) - Y(n-1)$ for $1 \le n \le 17$.

The encoder/decoder may also comprise of: means for computing $Y'(k) = Y(k) \cdot b_k$ for $0 \le k \le 17$;

means for computing $y'''(n) = \sum_{k=0}^{17} Y'(k) \cos[\frac{\pi}{2*18}(2k+1)n]$ for $0 \le n \le 17$;

means for computing $y'(n) = \begin{cases} y'''(n+9) & \text{for} \quad 0 \le n \le 8 \\ 0 & \text{for} \quad n = 9 \\ -y'''(27-n) & \text{for} \quad 10 \le n \le 26 \\ -y'''(n-27) & \text{for} \quad 27 \le n \le 35 \end{cases}$;

means for defining $y(0) = \sum_{k=0}^{18-1} Y(k) \cdot c_k$; and means for computing $y(n) = y'(n) - y(n-1)$ for $1 \le n \le 35$.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects, advantages and features of this invention will become more apparent from a consideration of the following detailed description and the drawings, in which:

FIG. 1 is an exemplary hardware butterfly diagram for a fast 18-18 DCT method;

FIG. 2 is an exemplary hardware butterfly diagram for a fast 36-18 MDCT method; and

FIG. 3 is an exemplary hardware butterfly diagram for fast 18-36 IMDCT described.

DEATILED DESCRIPTION

The system and method of the present invention computes the MDCT and its IMDCT, respectively with a substantially reduced number of multiplications and additions. Additionally, a computer simulation shows that the speed of these two new methods for computing the MDCT and IMDCT are 7.2 times and 4.3 times faster than the direct methods, respectively. The present invention significantly improves the performance and effectiveness of data compression methods such as, MP III, MPEG, and other audio/video compression methods. By this means, the MDCT or IMDCT defined in Hwang-Cheng

Chiang and Jie-Cherng Liu, "Regressive Implementation for the Forward and Inverse MDCT in MPEG Audio Coding, " *IEEE Signal Processing Letters,* vol.3, no.4, pp.116-117, April 1996 [10], the contents of which are hereby incorporated by reference, can be converted first into the standard DCT of length $N=18$. Then such an 18-point DCT can be implemented by using the same length Winograd's DFT method.

The advantage of this new method over the previous methods is that the 18-point DCT can be used to compute both the MDCT and IMDCT simultaneously. As a consequence, the computation of the 18-point IDCT needed to perform the MDCT method developed in Hsieh S. Hou, "A Fast Recursive Method for Computing the Discrete Cosine Transform," *IEEE Trans on Acoustics, Speech and Signal processing*, vol. ASSP-35, no. 10, pp. 1455-1461, Oct. 1987 [9], the contents of which are hereby incorporated by reference, is completely avoided in this new MDCT method. With these new techniques, the new MDCT and IMDCT methods require only a small fraction of the number of multiplications and additions that are required in direct methods, respectively.

Fast Method for Computing the 18-point DCT

Let $x(k)$ be a sequence with length $N=18$. The DCT of $x(k)$ defined in [3] is given by

$$X(n)=\sum_{k=0}^{18-1}x(k)\cos\frac{\pi}{36}(2k+1)n \qquad for \quad 0\le n\le 17 \qquad (1)$$

In (1), $X(n)$ can be decomposed into the even and odd indexed output of the DCT. That is,

$$A(n)=X(2n) \qquad for \quad 0\le n\le 8 \qquad (2)$$

$$B(n)=X(2n+1) \qquad for \quad 0\le n\le 8 \qquad (3)$$

By [3], (2) and (3) can be shown to be two DCT's with length 9 as fellows:

$$A(n)=\sum_{k=0}^{9-1}a(k)\cos[\frac{\pi}{2\times 9}(2k+1)n] \qquad for \quad 0\le n\le 8 \qquad (4)$$

where

$$a(k) = x(k) + x(18-1-k) \tag{5}$$

and

$$B'(n) = \sum_{k=0}^{9-1} b(k)\cos[\frac{\pi}{2\times 9}(2k+1)n] \qquad for \quad 0 \le n \le 8 \tag{6}$$

where

$$b(k) = 2[x(k) - x(18-1-k)]\cos[\frac{\pi}{2\times 18}(2k+1)] \tag{7}$$

$$B'(n) = B(n) + B(n-1) \qquad for \quad 0 \le n \le 8 \tag{8}$$

Note that $B(0) = B(-1)$. From (8), one obtains

$$B(0) = \frac{B'(0)}{2} \tag{9}$$

Using the sequence $B'(n)$ obtained by (6), $B(n)$ for $0 \le n \le 8$ can be obtained by the use of both (9) and the recursive form in (8).

Since $A(n)$ in (4) is a 9-point DCT, then, it is shown in [5] that this DCT can be computed by the use of the identical-length DFT method developed by Winograd [5,6]. To illustrate this, first, the decomposition of (4) into two even and odd values of n of the DCT appears as follows:

$$A(2n) = \sum_{k=0}^{9-1} a(k)\cos[\frac{\pi}{2\times 9}(2k+1)2n] \qquad for \quad 0 \le n \le 4 \tag{10}$$

and

$$A(9-2n) = \sum_{k=0}^{9-1} a(k)\cos[\frac{\pi}{2\times 9}(2k+1)(9-2n)] \quad for \quad 0 \le n \le 4 \tag{11}$$

It follows from [4] that (10) and (11) can be shown to the following:

$$A(2n) = (-1)^n \operatorname{Re}\{\sum_{k=0}^{9-1} a'(k)\omega^{nk}\} \tag{12}$$

and

$$A(9-2n) = (-1)^{n+1} \operatorname{Im}\{\sum_{k=0}^{9-1} a'(k)\omega^{nk}\} \tag{13}$$

where $\omega = e^{\frac{i2\pi}{9}}$ is the 9th root of unity and $a'(k)$ is defined by

$$a'(k) = \begin{cases} a((-1)^{k+6}k + 4) & for \quad 0 \le k \le 4 \quad (14) \\ a((-1)^{k+6}(9-k) + 4) & for \quad 5 \le k \le 8 \quad (15) \end{cases}$$

Define the 9-point DFT as follows:

$$A'(n) = \sum_{k=0}^{9-1} a'(k)\omega^{nk} \tag{16}$$

It is shown in [4] that the transform in (16) can be computed with a minimum number of operations by Winograd's method. The substitution of (16) into (14) and (13) yields

$$A(2n) = (-1)^n \operatorname{Re}\{A'(n)\} \qquad \text{for} \quad 0 \le n \le 4 \tag{17}$$

$$A(9-2n) = (-1)^{n+1} \operatorname{Im}\{A'(n)\} \qquad \text{for} \quad 5 \le n \le 8 \tag{18}$$

where $A'(n)$ is given in (16).

The detailed method given in [5] for computing the 9-point DCT in (4) is given in Appendix A. One observes from this method that the number of multiplications and additions are 10 and 34, respectively. In a similar fashion, by replacing $a(k)$ and $A(n)$ by $b(k)$ and $B'(n)$ respectively, (6) can also be computed by the use of the method given in Appendix A.

Let $x(k)$ be a sequence of length 18. The fast method for computing the 18-point DCT defined in (1) comprises the following three steps:

(1). Compute $a(k)$ and $b(k)$ from (5) and (7). That is,

$$a(k) = x(k) + x(18-1-k) \quad \text{for} \quad 0 \le k \le 8$$

$$b(k) = 2(x(k) - x(18-1-k))\cos[\frac{\pi}{36}(2k+1)] \quad \text{for} \quad 0 \le k \le 8.$$

(2). Use the fast method given in Appendix A to compute the 9-point DCT of $a(k)$, i.e. $A(n)$ in (4) and the 9-point DCT of $b(k)$, i.e., $B'(n)$ in (6) for $0 \le n \le 8$.

(3). Use both the recursive formulae for $1 \le n \le 8$ in (8) and (9) to compute $B(n)$ from the known $B'(n)$ for $0 \le n \le 8$. That is, $B(0) = B'(0)/2$, and $B(n) = B'(n) - B(n-1)$ for $1 \le n \le 8$. Then set $X(2n) = A(n)$, for $0 \le n \le 8$ And $X(2n+1) = B(n)$ for $0 \le n \le 8$, where $X(n)$ is a 18-point DCT of $x(k)$ given in (1).

From the method described above, it can be shown that the total number of multiplications and additions needed to compute (1) are 29 and 97, respectively. In contrast, 324 multiplications and 306 additions are required for a direct computation of (1). An exemplary butterfly diagram of a fast 18-18 DCT is depicted in FIG. 1. FIG. 1 shows an exemplary hardware butterfly diagram for the fast 18-18 DCT method. The button diagrams indicate the computation flows. The pre-computed constants are

$$d_k = 2\cos[\frac{\pi}{36}(2k+1)] \quad \text{for} \quad 0 \le k \le 8.$$

Fast method for Computing the MDCT

Let $y(k)$ be a sequence of length $N = 36$, the MDCT of $y(k)$ defined in [10] is given by

$$Y(n) = \sum_{k=0}^{36-1} y(k)\cos[\frac{\pi}{2*36}(2k+1+18)(2n+1)] \quad \text{for} \quad 0 \le n \le 17 \tag{19}$$

Instead of computing (19), an alternating technique is employed. To see this, replacing k by $36-1-k$ in (19), one obtains

$$Y(n) = \sum_{k=0}^{35} y(36-1-k)\cos[\frac{\pi}{2*36}((2*36)(2n+1)-(2k+1-18)(2n+1))] \tag{20}$$

But

$$\cos[\frac{\pi}{2*36}((2*36(2n+1)-(2k+1-18)(2n+1))] = -\cos[\frac{\pi}{2*36}(2k+1-18)(2n+1)]$$

Thus, (19) becomes

$$Y(n) = -\sum_{k=0}^{35} y(36-1-k)\cos[\frac{\pi}{2*36}(2k+1-18)(2n+1)] \quad \text{for} \quad 0 \le n \le 17 \tag{21}$$

If

$$Y'(n) = Y(n) + Y(n-1) \quad \text{for} \quad 0 \le n \le 17 \tag{22}$$

then

$$'(n) = \sum_{k=0}^{35} y'(k)\cos[\frac{\pi}{2*36}(2k+1-18)(2n)] \tag{23a}$$

where

$$y'(k) = -2y(36-1-k)\cos[\frac{\pi}{2*36}(2k+1-18)] \quad \text{for} \quad 0 \le k \le 35 \tag{23b}$$

The substitution of (21) into (22) yields,

$$\begin{aligned} Y'(n) &= -\sum_{k=0}^{36-1} y(36-1-k)\cos[\frac{\pi}{2*36}(2k+1-18)(2n+1)] \\ &\quad -\sum_{k=0}^{36-1} y(36-1-k)\cos[\frac{\pi}{2*36}(2k+1-18)(2n-1)] \\ &= -\sum_{k=0}^{35} y(36-1-k)\cos[\frac{\pi}{2*36}((2k+1-18)(2n)+(2k+1-18))] \\ &\quad -\sum_{k=0}^{35} y(36-1-k)\cos[\frac{\pi}{2*36}((2k+1-18)(2n)-(2k+1-18))] \end{aligned} \tag{24}$$

(24) can be shown as

$$\begin{aligned} Y'(n) &= -\sum_{k=0}^{35} 2y(36-1-k)\cos[\frac{\pi}{2*36}(2k+1-18)]\cos[\frac{\pi}{2*36}(2k+1-18)(2n)] \\ &= \sum_{k=0}^{35} y'(k)\cos[\frac{\pi}{36}(2k+1-18)n] \end{aligned} \tag{25}$$

where $y'(k)$ is given in (23b).

From (21), one observes that $Y(0) = Y(-1)$. Using this result and the recursive formula in (22), one yields

$$Y(0) = Y'(0)/2 \tag{26a}$$

$$Y(n) = Y'(n) - Y(n-1) \quad \text{for} \quad 1 \le n \le 17 \tag{26b}$$

Using the sequence $y'(n)$ obtained by (25b), the sequence $y(n)$ for $0 \le n \le 17$ can be obtained by the use of (26a) and (26b).

Let

$$y''(k) = \begin{cases} y'(k+9) & \text{for} \quad 0 \le k \le 26 \\ y'(k-27) & \text{for} \quad 27 \le k \le 35 \end{cases} \tag{27a}$$

(27b)

Then

$$Y'(n) = \sum_{k=0}^{35} y''(k) \cos[\frac{\pi}{36}(2k+1)n] \tag{28}$$

The proof of this is similar to that used in Lemma 1 in [7]. To prove this, (25) can be rewritten as:

$$Y'(n) = \sum_{k=0}^{8} y'(k) \cos[\frac{\pi}{36}(2k+1-18)n] + \sum_{k=9}^{35} y'(k) \cos[\frac{\pi}{36}(2k+1-18)n] \tag{29}$$

Let $k = k'+9$, (29) becomes

$$Y'(n) = \sum_{k'=-9}^{-1} y'(k'+9) \cos[\frac{\pi}{36}(2k'+1)n] + \sum_{k=0}^{26} y'(k'+9) \cos[\frac{\pi}{36}(2k'+1)n] \tag{30}$$

Also, let $k'+9 = k-27$ in the first summation and let $k' = k$ in the second summation of (30). Then

$$Y'(n) = \sum_{k=27}^{35} y'(k-27) \cos[\frac{\pi}{36}((2k+1)n - (2\times 36)n)] + \sum_{k=0}^{26} y'(k+9) \cos[\frac{\pi}{36}(2k+1)n] \tag{31}$$

But

$$\cos[\frac{\pi}{36}((2k+1)n - (2\times 36)n)] = \cos[\frac{\pi}{36}(2k+1)n]$$

Thus, (31) becomes

$$Y'(n) = \sum_{k=0}^{35} y''(n) \cos[\frac{\pi}{36}(2k+1)n] \tag{32}$$

where $y''(k)$ is given in (27a) and (27b).

For given $y''(k)$ as defined in (27a) and (27b),

$$X(n) = Y'(n) = \sum_{k=0}^{17} x(k)\cos[\frac{\pi}{36}(2k+1)n] \quad \text{for } 0 \le n \le 17 \tag{33}$$

where

$$x(k) = y''(k) + y''(36-1-k) \quad \text{for } 0 \le k \le 1 \tag{34}$$

The proof of this is also similar to that used in Lemma 2 in [7]. To see this, (32) can be rewritten as

$$Y'(n) = Y_1'(n) + Y_2'(n) \quad \text{for } 0 \le n \le 1 \tag{35}$$

where

$$Y_1'(n) = \sum_{k=0}^{17} y''(k)\cos[\frac{\pi}{36}(2k+1)n] \tag{36}$$

and

$$Y_2'(n) = \sum_{k=18}^{35} y''(k)\cos[\frac{\pi}{36}(2k+1)n] \tag{37}$$

Let $k = 36-1-k'$. Then $Y_2'(n)$ in (37) becomes

$$Y_2'(n) = \sum_{k'=17}^{0} y''(36-1-k')\cos[\frac{\pi}{36}(2(36-1-k')+1)n] \tag{38}$$

However,

$$\cos\frac{\pi}{36}(2\times 36n - (2k'+1)n) = \cos\frac{\pi}{36}(2k'+1)n$$

Hence, let $k' = k$, (38) becomes

$$Y_2'(n) = \sum_{k'=0}^{17} y''(36-1-k')\cos\frac{\pi}{36}(2k'+1)n \tag{39}$$

The substitution (39) and (36) into (35) yields

$$'(n) = \sum_{k=0}^{17} y''(k)\cos[\frac{\pi}{36}(2k+1)n] + \sum_{k=0}^{17} y''(36-1-k)\cos[\frac{\pi}{36}(2k+1)n]$$
$$= \sum_{k=0}^{17} x(k)\cos[\frac{\pi}{36}(2k+1)n] \quad \text{for } 0 \le n \le 17 \tag{40}$$

where $x(k)$ is given in (33). From the above, it can be seen that (33) is an 18-point DCT which can be efficiently computed by the use of the fast method given in the previous section.

Suppose that $y(k)$ is a sequence of length $N = 36$. For $0 \le k \le 8$, one has $27 \le 36-1-k \le 35$. Thus, from (27) and (23b), (34) becomes

$$\begin{aligned} x(k) &= y''(k) + y''(36-1-k) \\ &= y'(k+9) + y'(8-k) \\ &= -2y(26-k)\cos[\frac{\pi}{2\times 36}(2k+1)] - 2y(27+k)\cos[\frac{\pi}{2\times 36}(2k+1)] \\ &= [-y(26-k) - y(27+k)]\cdot 2\cos[\frac{\pi}{2\times 36}(2k+1)] \end{aligned}$$

Similar, for $9 \le k \le 17$, (34) becomes

$$\begin{aligned} x(k) &= y'(k+9) + y'(44-k) \\ &= -2y(26-k)\cos[\frac{\pi}{2\times 36}(2k+1)] - 2y(k-9)\cos[\frac{\pi}{2\times 36}(71-2k)] \\ &= [y(k-9) - y(26-k)]\cdot 2\cos[\frac{\pi}{72}(2k+1)] \end{aligned}$$

since $\cos[\frac{\pi}{2\times 36}(71-2k)] = -\cos[\frac{\pi}{2\times 36}(2k+1)]$.

Let $b_k = 2\cos[\frac{\pi}{2\times 36}(2k+1)]$ be the pre-computed constants. The fast method for computing the MDCT of $y(k)$ in (19) is composed of the following three successive steps of

computation:

(1). Compute

$$x(k) = \begin{cases} [-y(26-k) - y(27+k)] \cdot b_k & \text{for} \quad 0 \le k \le 8 \\ [y(k-9) - y(26-k)] \cdot b_k & \text{for} \quad 9 \le k \le 17 \end{cases}.$$

(2). Use the fast method described in Section II to compute the 18-point DCT of $x(k)$, i.e., $Y'(n)$ defined in (40). That is, $Y'(n) = \sum_{k=0}^{17} x(k) \cos[\frac{\pi}{36}(2k+1)n]$ for $0 \le n \le 17$

(3). Let $Y(0) = Y'(0)/2$.

Then compute $Y(n) = Y'(n) - Y(n-1)$ for $1 \le n \le 17$.

This new method is simpler and faster than that of the brute-force method. The number of multiplications and additions needed for this method and the brute-force method is given in Table 1. Table 1 shows that the multiplications and additions needed to implement this new method require only 10.03% and 20.95% of the brute-force method, respectively. The butterfly diagram of the fast 36-18 MDCT is depicted in FIG. 2. FIG. 2 illustrates an exemplary hardware butterfly diagram for the above fast 36-18 MDCT method. The button diagram indicates the computation flow. The pre-computed constants are

$$b_k = 2\cos[\frac{\pi}{2 \times 36}(2k+1)].$$

Fast Method for Computing the IMDCT

Let $Y(k)$ for $0 \le k \le 17$ be the output sequence of (19). The inverse MDCT of $Y(k)$

defined in [10] is given by

$$y(n)=\sum_{k=0}^{18-1}Y(k)\cos[\frac{\pi}{2*36}(2n+1+18)(2k+1)] \quad \text{for} \quad 0\le n\le 35 \tag{41}$$

If $y'(n)=y(n)+y(n-1)$, using a proof similar to that used in (24)-(26), then (41) can be expressed by

$$y'(n)=\sum_{k=0}^{18-1}Y'(k)\cos[\frac{\pi}{2*36}(2n+18)(2k+1)] \quad \text{for} \quad 0\le n\le 35 \tag{42}$$

where

$$Y'(k)=2Y(k)\cos[\frac{\pi}{2*36}(2k+1)] \quad \text{for} \quad 0\le k\le 17 \tag{43}$$

It follows from (41) that

$$y(0)=\sum_{k=0}^{18-1}Y(k)\cos[\frac{\pi}{2*36}(19)(2k+1)] \tag{44}$$

Using the sequence $y'(n)$ obtained by (42), again, $y(n)$ can be obtained by the use of the initial condition $y(0)$ given in (44) and the recursive formula $y(n)=y'(n)-y(n-1)$ for $1\le n\le 35$.

Define

$$y''(n)=\begin{cases} y'(n-9) & \text{for} \quad 9\le n\le 35 \quad (45)\\ y'(n+27) & \text{for} \quad 0\le n\le 8 \quad (46)\end{cases}$$

From (42), (45) and (46) can be shown to the following

$$y''(n)=y'(n-9)=\sum_{k=0}^{18-1}Y'(k)\cos[\frac{\pi}{2*36}(2k+1)2n] \quad \text{for} \quad 9\le n\le 35 \tag{47}$$

$$y''(n) = y'(n+27) = \sum_{k=0}^{18-1} Y'(k)\cos[\frac{\pi}{2*36}(2n+72)(2k+1)] \quad \text{for} \quad 0 \le n \le 8 \qquad (48)$$

In (47), it is can be shown that $y''(18) = 0$. Based on [7], the following can be proved:

In (47) and (48),

$$y''(18+j) = -y''(18-j) \qquad \text{for} \quad 1 \le j \le 9 \qquad (49)$$

$$y''(18+j) = y''(18-j) \qquad \text{for} \quad 10 \le j \le 17 \qquad (50)$$

Thus, for $1 \le j \le 9$, from (47), one has

$$\begin{aligned} y''(18+j) &= \sum_{k=0}^{18-1} Y'(k)\cos[\frac{\pi}{2*36}\cdot 2(18+j)(2k+1)] \\ &= \sum_{k=0}^{18-1} Y'(k)\cos[\frac{\pi}{2*36}(2*18)(2k+1) + \frac{\pi}{4*18}(2j)(2k+1)] \end{aligned} \qquad (51)$$

Using $\cos(A+B) = \cos A \cdot \cos B - \sin A \cdot \sin B$, (51) becomes

$$y''(18+j) = -\sum_{k=0}^{18-1} Y'(k)\sin[\frac{\pi}{2}(2k+1)]\cdot\sin[\frac{\pi j}{2*18}(2k+1)] \text{ for } 1 \le j \le 9 \qquad (52)$$

Similarly,

$$\begin{aligned} y''(18-j) &= \sum_{k=0}^{18-1} Y'(k)\cos[\frac{\pi}{4*18}2(18-j)(2k+1)] \\ &= \sum_{k=0}^{18-1} Y'(k)\sin[\frac{\pi}{2}(2k+1)]\cdot\sin[\frac{\pi j}{2*18}(2k+1)] \quad \text{for} \quad 1 \le j \le 9 \end{aligned} \qquad (53)$$

From (52) and (53), $y''(18+j) = -y''(18-j)$ for $1 \le j \le 9$.

For $10 \le j \le 17$, from (48), one has

$$y''(18-j) = \sum_{k=0}^{17} Y'(k)\cos[\frac{\pi}{2*36}(2(18-j)+72)(2k+1)]$$
$$= \sum_{k=0}^{17} Y'(k)\cos[\frac{108\pi}{2*36}(2k+1) - \frac{\pi j}{2*18}(2k+1)] \quad (54)$$

Again, using $\cos(A-B) = \cos A \cdot \cos B + \sin A \cdot \sin B$, (54) becomes

$$y''(18-j) = \sum_{k=0}^{17} Y'(k)\sin[\frac{3\pi}{2}(2k+1)\cdot \sin\frac{\pi j}{2*18}(2k+1)] \quad (55)$$

However, $\sin\frac{3\pi}{2}(2k+1) = -\sin\frac{\pi}{2}(2k+1)$. Hence, (55) becomes

$$y''(18-j) = -\sum_{k=0}^{17} Y'(k)\sin[\frac{\pi}{2}(2k+1)]\cdot\sin[\frac{\pi j}{2*18}(2k+1)] \quad (56)$$

From (52) and (54), $y''(18+j) = -y''(18-j)$ for $10 \le j \le 17$.

It follows from the above proof that one only computes $y''(n)$ for $0 \le n \le 17$. In other words, for given $y''(n)$, where $0 \le n \le 17$, $y''(n)$ for $19 \le n \le 35$ can be obtained from using (49) and (50). Recall that $y''(18) = 0$.

Also, by defining

$$y'''(n) = -y''(n) \qquad \text{for} \quad 0 \le j \le 8 \quad (57)$$

T.0184

$$y'''(n) = y''(n) \qquad \text{for} \quad 9 \le j \le 17 \quad (58)$$

Then

T.0185

$$y'''(n) = \sum_{k=0}^{17} Y'(k)\cos[\frac{\pi}{2*18}(2k+1)n] \qquad \text{for} \quad 0 \le n \le 17 \quad (59)$$

As a result, for $0 \le n \le 8$, from (48), one yields

$$y''(n) = \sum_{k=0}^{17} Y'(k)\cos[\frac{\pi}{2*18}n(2k+1)+\pi(2k+1)]$$
$$= -\sum_{k=0}^{17} Y'(k)\cos[\frac{\pi}{2*18}(2k+1)n]$$
$$= -y'''(x)$$

For $9 \le n \le 17$, from (47), one obtains $y'''(n) = y''(n)$.

In (59), $y'''(n)$ is an 18-point DCT. By replacing $y'''(n)$ and $Y'(k)$ by $X(n)$ and $x(k)$, respectively, the 18-point DCT in (59) is the same as (1). As a consequence, the fast method developed in Section II can also be used to compute (59).

From (45) and (46), one has

$$y'(n) = \begin{cases} y''(n+9) & \text{for} \quad 0 \le n \le 26 \\ y''(n-27) & \text{for} \quad 27 \le n \le 35 \end{cases}$$

Together with (49), (50), (57) and (58), $y'(n)$ can be expressed in terms of $y'''(n)$ as follows:

$$y'(n) = y''(n+9) = y'''(n+9) \qquad \text{for } 0 \le n \le 8$$

$$y'(n) = y''(n+9) = y''(18) = 0 \qquad \text{for } n = 9$$

$$y'(n) = y''(n+9)$$
$$= y''(18+(n-9))$$
$$= -y''(18-(n-9)) \qquad \text{for } 10 \le n \le 18$$
$$= -y''(27-n)$$
$$= -y'''(27-n)$$

$$y'(n) = -y'''(27-n) \qquad \text{for } 19 \le n \le 26$$

$$y'(n) = -y'''(n-27) \qquad \text{for } 27 \le n \le 35$$

Let $Y(k)$ for $0 \le k \le 17$ be the output sequence of (19). Also let $b_k = 2\cos[\frac{\pi}{2\times 36}(2k+1)]$ be the pre-computed constants which are the same as those defined in the MDCT method in the previous section. Finally, let $c_k = \cos[\frac{19\pi}{2\times 36}(2k+1)]$.

The fast method for computing (41) is comprised of the following 4 steps of computation:

(1). Compute

$$Y'(k) = Y(k)\cdot b_k \quad \text{for} \quad 0 \le k \le 17$$

(2). Use the fast method described in Section II to compute the 18-point DCT of $Y(k)$ given in (59). That is,

$$y'''(n) = \sum_{k=0}^{17} Y'(k)\cos[\frac{\pi}{2*18}(2k+1)n] \quad \text{for} \quad 0 \le n \le 17$$

(3). Compute

$$y'(n) = \begin{cases} y'''(n+9) & \text{for} \quad 0 \le n \le 8 \\ 0 & \text{for} \quad n = 9 \\ -y'''(27-n) & \text{for} \quad 10 \le n \le 26 \\ -y'''(n-27) & \text{for} \quad 27 \le n \le 35 \end{cases}$$

(4). Let $y(0) = \sum_{k=0}^{18-1} Y(k)\cdot c_k$.

Then compute $y(n) = y'(n) - y(n-1)$ for $1 \le n \le 35$.

It is easy to observe that this new method for computing the IMDCT is simpler and faster than that of the brute-force method. The number of multiplications and additions needed for this fast method and the brute-force method is given Table 1. Table 1 shows

that the number of multiplications and additions required to implement this new method are only 7.25% and 10.03%, respectively, of the brute-force method. The butterfly diagram of the fast 18-36 IMDCT is depicted in Figure 3.

Program Implementation and Simulation Results

The MDCT and its IMDCT methods described above can be computed simultaneously by using the 18-point DCT. These two new methods are implemented using C++ language run on a Pentium 133/Windows 98 platform, in which the multiplications and additions are implemented using double data type. An example of the source code is given in Appendix B. The fast methods are verified by comparing their results of 10^6 computations to those of the bruit-force methods. The computation times are provided in Table 1, which is in millisecond averaged from 10^6 computations. These new methods for computing the MDCT and IMDCT require 0.0218 ms and 0.0375 ms as compared to 0.1567 ms and 0.1616 ms required by the brute-force methods, respectively.

As a consequence, the new methods for computing the MDCT and IMDCT are 7.2 and 4.3 times faster than the brute-force methods, respectively. In addition to audio compression such as MP III method, the resulting improvement is also useful for video compression methods such as MPEG method.

Additionally, the method of the present invention may be implemented in a custom application-specific integrated circuits (ASICs), general-purpose programmable DSP using firmware to program the DSP devices, and customizable arrays such as programmable logic arrays (PLAs) and field-programmable arrays (FPAs). Because the present invention uses the same encoding and decoding (recording and playback), the complexity of software, firmware, and hardware is significantly reduced resulting in cost improvement and flexibility, in addition to speed improvement.

FIG. 3 is an exemplary hardware butterfly diagram for fast 18-36 IMDCT described. The button diagram indicates the computation flow. The pre-computed constants are

$b_k = 2\cos[\frac{\pi}{2\times 36}(2k+1)]$ and

$$y(0) = \sum_{k=0}^{18-1} Y(k) \cdot c_k \text{ where } c_k = \cos[\frac{19\pi}{2 \times 36}(2k+1)].$$

Table I: NUMBER OF MULTIPLICATIONS AND ADDITIONS NEEDED TO IMPLEMENT THE MDCT AND IMDCT

		MDCT				IMDCT			
		Step 1	Step 2	Step 3	Total	Step 1	Step 2	Step 3,4	Total
New Method	Multipliers	18	29	0	47	18	29	18	65
	Adders	18	94	17	129	0	94	35	129
Brute-force method	Multipliers				648				648
	Adders				630				612

Table II: MDCT AND IMDCT EXECUTION TIMES IN MILLI SECOND

	MDCT	IMDCT
New method	0.0218	0.0375
Brute-force methods	0.1567	0.1616

It will be recognized by those skilled in the art that various modifications may be made to the illustrated and other embodiments of the invention described above, without departing from the broad inventive scope thereof. It will be understood therefore that the invention is not limited to the particular embodiments or arrangements disclosed, but is rather intended to cover any changes, adaptations or modifications which are within the scope and spirit of the claims in the area of signal processing of audio and video signals including compression of audio and video signals.

Appendix A:

This appendix contains the method developed in [5] for computing the 9-point DCT given by

$$A(n) = \sum_{k=0}^{9-1} a(k) \cos\left[\frac{\pi}{2 \times 9}(2k+1)n\right] \quad \text{for} \quad 0 \le n \le 8$$

Method for the 9-point DCT:

$c_1 = -0.866025, c_2 = 0.939693, c_3 = -0.173648, c_4 = -0.766044$
$c_5 = 0.5, c_6 = -0.342020, c_7 = -0.984808, c_8 = -0.642788$
$d_1 = a(3) + a(5), d_2 = a(3) - a(5), d_3 = a(6) + a(2), d_4 = a(6) - a(2)$
$d_5 = a(1) + a(7), d_6 = a(1) - a(7), d_7 = a(8) + a(0), d_8 = a(8) - a(0)$
$d_9 = a(4) + d_5, d_{10} = d_1 + d_3, d_{11} = d_{10} + d_7, d_{12} = d_3 - d_7, d_{13} = d_1 - d_7, d_{14} = d_1 - d_3$
$d_{15} = d_2 - d_4, d_{16} = d_{15} + d_8, d_{17} = d_4 + d_8, d_{18} = d_2 - d_8, d_{19} = d_2 + d_4$
$m_1 = c_1 d_6, m_2 = c_5 d_5, m_3 = c_5 d_{11}, m_4 = c_2 d_{12}, m_5 = c_3 d_{13}$
$m_6 = c_4 d_{14}, m_7 = c_1 d_{16}, m_8 = c_6 d_{17}, m_9 = c_7 d_{18}, m_{10} = c_8 d_{19}$
$d_{20} = a(4) - m_2, d_{21} = d_{20} + m_4, d_{22} = d_{20} - m_4, d_{23} = d_{20} + m_5$
$d_{24} = m_1 + m_8, d_{25} = m_1 - m_8, d_{26} = m_1 + m_9$
$A_0 = d_9 + d_{11}, \mathrm{A}_1 = m_{10} - d_{26}, \mathrm{A}_2 = m_6 - d_{21}, A_3 = m_7, A_4 = d_{22} - m_5$
$A_5 = d_{25} - m_9, \mathrm{A}_6 = m_3 - d_9, \mathrm{A}_7 = d_{24} + m_{10}, A_8 = d_{23} + m_6$

Thus, the 9-point DCT requires only 10 multiplications and 34 additions, respectively.

Appendix B: Source Code

```
//*****************************************
// FIle name: mdct.cpp
//*****************************************


#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include <time.h>

#pragma hdrstop
#include <condefs.h>
#include "dct_lib.h"

USEUNIT("dct_lib.cpp");
//---------------------------------------
int borg[100036];
double
bin[100036],*ptbin,bout0[36],bout1[36];

//---------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int k,n;
int i,cnt;
double diff;
clock_t ck1,ck2;

cout<<"welcome to mdct test\n";
build_costab();

// the random data
randomize();
for(k=0;k<100036;k++)
borg[k]=random(255);
for(k=0;k<100036;k++)      bin[k]=(double)
borg[k];


// verify the MDCT and the fast methods
cnt=100000;
ptbin=bin;
cout<<"Verifying MDCT3618 and MDCT3618F,
cnt="<<cnt<<endl<<endl;
for(i=0;i<cnt;i++)
     {
     MDCT3618 (ptbin,bout0);
     MDCT3618F(ptbin,bout1);
     ptbin++;
     for(n=0;n<18;n++)
          {
                              diff=((bout0[n]-
bout1[n])<.0000000001)?0:99999;
          if(diff>0)
cout<<diff<<setw(10)<<bout0[n]<<setw(10)<
<bout1[n]<<"\n";
          }
     }
// verify the IMDCT and the fast methods
ptbin=bin;
cout<<"Verifying       IMDCT1836       and
IMDCT1836F, cnt="<<cnt<<endl<<endl;
for(i=0;i<cnt;i++)
     {
     IMDCT1836 (ptbin,bout0);
     IMDCT1836F(ptbin,bout1);
     ptbin++;
     for(n=0;n<36;n++)
          {
                              diff=((bout0[n]-
bout1[n])<.0000000001)?0:99999;
          if(diff>0)
cout<<diff<<setw(10)<<bout0[n]<<setw(10)<<bo
ut1[n]<<"\n";
          }
     }


cnt=100000;
// computation time of MDCT3618
cout<<"Computation time of MDCT3618    : ";
ptbin=bin;
ck1=clock();
for(i=0;i<cnt;i++)
     {
     MDCT3618 (ptbin,bout0);
     ptbin++;
     }
ck2=clock();
cout<<setw(6)<<(ck2-ck1)<<endl;


// Computation time of MDCT3618F
cout<<"Computation time of MDCT3618F   : ";
ptbin=bin;
ck1=clock();
for(i=0;i<cnt;i++)
     {
     MDCT3618F(ptbin,bout1);
     ptbin++;
     }
ck2=clock();
cout<<setw(6)<<(ck2-ck1)<<endl;


//Computation time of IMDCT1836
cout<<"Computation time of IMDCT1836   : ";
ptbin=bin;
ck1=clock();
for(i=0;i<cnt;i++)
     {
     IMDCT1836 (ptbin,bout0);
     ptbin++;
     }
ck2=clock();
cout<<setw(6)<<(ck2-ck1)<<endl;


// Computation time of IMDCT1836F
cout<<"Computation time of IMDCT1836F : ";
ptbin=bin;
ck1=clock();
for(i=0;i<cnt;i++)
```

```
    {
    IMDCT1836F(ptbin,bout1);
    ptbin++;
    }
ck2=clock();
cout<<setw(6)<<(ck2-ck1)<<endl<<endl;

return 0;
}
//***************************************
// FIle name: dct_lib.cpp
//***************************************



#include <iostream.h>
#include <math.h>
#pragma hdrstop

#include "dct_lib.h"

//---------------------------------------
#pragma package(smart_init)

// some pre-computed consine tables
double Cos99[9][9];
double Cos99W[9];   // used for DCT99W
double Cos1818[18][18];
double Cos1818F[18];// DCT1818F
double Cos3618[18][36];  //   shared   for
mdct3618, imdct1836
double Cos3618F[18];
double Cos1836F[18];


//=======================================
// the COS tables for various size
//=======================================
void build_costab(void)
{
int n,k;

// DCT99
for(n=0;n<9;n++)
     for(k=0;k<9;k++)
     Cos99[n][k]=cos(M_PI/18*(2*k+1)*n);

// for DCT99W
Cos99W[1]=Cos99[1][7];
Cos99W[2]=Cos99[2][0];
Cos99W[3]=Cos99[2][2];
Cos99W[4]=Cos99[2][3];
Cos99W[5]=Cos99[2][1];
Cos99W[6]=Cos99[1][5];
Cos99W[7]=Cos99[1][8];
Cos99W[8]=Cos99[1][6];


// DCT1818
for(n=0;n<18;n++)
     for(k=0;k<18;k++)

     Cos1818[n][k]=cos(M_PI/36*(2*k+1)*n)
;
// DCT1818F
for(k=0;k<18;k++)
     Cos1818F[k]=2*Cos1818[1][k];

// DCT3618
for(n=0;n<18;n++)
     for(k=0;k<36;k++)

     Cos3618[n][k]=cos(M_PI/72*(2*k+19)*(2*n
+1));

// DCT3618F
for(k=0;k<18;k++)
     Cos3618F[k]=2*cos(M_PI/72*(2*k+1));

// IMDCT1836F
for(k=0;k<18;k++)
     Cos1836F[k]=cos(M_PI/72*19*(2*k+1));
}
//=======================================



//=======================================
// 9-9 DCT, standard
//=======================================
void DCT99(double *a, double *A)
{
int n,k;
double s;

for(n=0;n<9;n++)
     {
    s=0;
    for(k=0;k<9;k++)
     {
         s+=a[k]*Cos99[n][k];
        }
    A[n]=s;
    }
}



//=======================================
// 9-9 DCT, Winogard
//=======================================
void DCT99W(double *a, double *A)
{
double d[64],m[64];

d[1]=a[3]+a[5];
d[2]=a[3]-a[5];
d[3]=a[6]+a[2];
d[4]=a[6]-a[2];
d[5]=a[1]+a[7];
d[6]=a[1]-a[7];
d[7]=a[8]+a[0];
d[8]=a[8]-a[0];

d[ 9]=a[ 4]+d[5];
d[10]=d[ 1]+d[3];
d[11]=d[10]+d[7];
d[12]=d[ 3]-d[7];
d[13]=d[ 1]-d[7];
```

```
d[14]=d[ 1]-d[3];
d[15]=d[ 2]-d[4];
d[16]=d[15]+d[8];
d[17]=d[ 4]+d[8];
d[18]=d[ 2]-d[8];
d[19]=d[ 2]+d[4];

m[ 1]=Cos99W[1]*d[ 6];
m[ 2]=Cos99W[5]*d[ 5];
m[ 3]=Cos99W[5]*d[11];
m[ 4]=Cos99W[2]*d[12];
m[ 5]=Cos99W[3]*d[13];
m[ 6]=Cos99W[4]*d[14];
m[ 7]=Cos99W[1]*d[16];
m[ 8]=Cos99W[6]*d[17];
m[ 9]=Cos99W[7]*d[18];
m[10]=Cos99W[8]*d[19];

d[20]=a[ 4]-m[2];
d[21]=d[20]+m[4];
d[22]=d[20]-m[4];
d[23]=d[20]+m[5];
d[24]=m[ 1]+m[8];
d[25]=m[ 1]-m[8];
d[26]=m[ 1]+m[9];

A[0]=d[ 9]+d[11];
A[1]=m[10]-d[26];
A[2]=m[ 6]-d[21];
A[3]=m[ 7];
A[4]=d[22]-m[ 5];
A[5]=d[25]-m[ 9];
A[6]=m[ 3]-d[ 9];
A[7]=d[24]+m[10];
A[8]=d[23]+m[ 6];
}
//=======================================



//=======================================
// 18-18 DCT, standard
//=======================================
void DCT1818(double *a, double *A)
{
int n,k;
double s;

for(n=0;n<18;n++)
     {
    s=0;
    for(k=0;k<18;k++)
     {
         s+=a[k]*Cos1818[n][k];
        }
    A[n]=s;
    }
}
//=======================================



//=======================================
// 18-18 DCT, fast
//=======================================
void DCT1818F(double *x, double *X)
{
int n,k;
double a[9],b[9],A[9],B[9],Bp[9];

for(k=0;k<9;k++)
     {
    a[k]=x[k]+x[18-1-k];
    b[k]=(x[k]-x[18-1-k])*Cos1818F[k];
    }
DCT99W(a,A);
DCT99W(b,Bp);

B[0]=Bp[0]/2;
for(n=1;n<9;n++) B[n]=Bp[n]-B[n-1];

for(n=0;n<9;n++)
     {
    X[2*n]=A[n];
    X[2*n+1]=B[n];
    }
}
//=======================================



//=======================================
// 36-18 DCT, standard
//=======================================
void MDCT3618(double *a, double *A)
{
int n,k;
double s;

for(n=0;n<18;n++)
     {
    s=0;
    for(k=0;k<36;k++)
     {
         s+=a[k]*Cos3618[n][k];
        }
    A[n]=s;
    }
}
//=======================================



//=======================================
// 36-18 DCT, fast
//=======================================
void MDCT3618F(double *y, double *Y)
{
int n,k;
double x[18],Yp[18];

for(k=0;k<9;k++)
    x[k]=(-y[26-k]-y[27+k])*Cos3618F[k];
for(k=9;k<18;k++)
    x[k]=( y[k-9 ]-y[26-k])*Cos3618F[k];

DCT1818F(x,Yp);
Y[0]=Yp[0]/2;
for(n=1;n<18;n++) Y[n]=Yp[n]-Y[n-1];
}
```

```
//=======================================


//=======================================
// 18-36 IMDCT, standard
//=======================================
void IMDCT1836(double *Y, double *y)
{
int n,k;
double s;

for(n=0;n<36;n++)
     {
    s=0;
    for(k=0;k<18;k++)
     {
             s+=Y[k]*Cos3618[k][n];//share
with the same table as 3618
        }
    y[n]=s;
    }
}
//=======================================



//=======================================
// 36-18 DCT, fast
//=======================================
void IMDCT1836F(double *Y, double *y)
{
int n,k;
double Yp[18],yppp[18],yp[36],s;

for(k=0;k<18;k++)
    Yp[k]=Y[k]*Cos3618F[k];//the      same
table

DCT1818F(Yp,yppp);

for(n= 0;n< 9;n++) yp[n]= yppp[n+9];
for(n= 9;n<10;n++) yp[n]= 0;
for(n=10;n<27;n++) yp[n]=-yppp[27-n];
for(n=27;n<36;n++) yp[n]=-yppp[n-27];

s=0;
//for(k=0;k<18;k++) s+=Yp[k];
for(k=0;k<18;k++) s+=Y[k]*Cos1836F[k];
y[0]=s;
for(n=1;n<36;n++) y[n]=yp[n]-y[n-1];
}
//=======================================
```

```
//****************************************//
FIle name: dct_lib.h
//****************************************


#ifndef lib_dctH
#define lib_dctH
//=======================================
void build_costab(void);
void DCT99 (double *a, double *A);
void DCT99W(double *a, double *A);
void DCT1818 (double *a, double *A);
void DCT1818F(double *a, double *A);
void MDCT3618 (double *a, double *A);
void MDCT3618F(double *a, double *A);
void IMDCT1836 (double *Y, double *y);
void IMDCT1836F(double *Y, double *y);
#endif
```